Balanced Fair Resource Sharing in Computer Clusters

Thomas Bonald, Céline Comte

Balanced Fair Resource Sharing in Computer Clusters

Thomas Bonald[a], Céline Comte[b,a,1,*]

[a] *Télécom ParisTech, Université Paris-Saclay, France*
[b] *Nokia Bell Labs, France*

Abstract

We represent a computer cluster as a multi-server queue with some arbitrary graph of compatibilities between jobs and servers. Each server processes its jobs sequentially in FCFS order. The service rate of a job at any given time is the sum of the service rates of all servers processing this job. We show that the corresponding queue is quasi-reversible and use this property to design a scheduling algorithm achieving balanced fair sharing of the computing resources.

Keywords: Parallel processing, multi-server queues, balanced fairness, order independent queues, Whittle networks
2010 MSC: 60K25 Queueing theory, 68M20 Performance evaluation; queueing; scheduling

1. Introduction

Load balancing is a critical component of large-scale computer clusters. The flow of requests must be directed to the servers under various constraints like data availability, state of the servers and service level agreements. In this paper, we represent these constraints by an arbitrary graph of compatibilities between jobs and servers. The computer cluster can then be viewed as a multi-server queue where jobs are allocated to servers according to this graph. We assume that each server processes its jobs sequentially in FCFS order. The service rate of a job at any given time is the sum of the service rates of all servers processing this job, which means that resource pooling does not induce any processing overhead. We prove that, for Poisson job arrivals and exponential job sizes, this multi-server queue is quasi-reversible [13]. Exploiting this property, we design a novel scheduling algorithm achieving *balanced fair* sharing of the computing resources. This makes the stationary distribution of the system state insensitive to the job size distribution beyond the mean, a practically interesting property leading to simple and robust engineering rules.

*Corresponding author
Email addresses: thomas.bonald@telecom-paristech.fr (Thomas Bonald), celine.comte@nokia.com (Céline Comte)
[1] The authors are members of LINCS, see http://www.lincs.fr.

Balanced fairness was introduced in the context of data networks as the most efficient resource allocation having the insensitivity property, allowing the service provider to develop dimensioning rules based on average traffic only, and not on detailed traffic characteristics [5]. Formally, it is the only allocation such that the underlying Markov process is reversible and at least one resource is saturated in each state. Balanced fairness has later been used to evaluate the performance of content-distribution networks [22, 23]. However, no scheduling algorithm has been proved so far to achieve this allocation, except in some specific cases where it coincides with proportional fairness [14, 27]. To the best of our knowledge, our scheduling algorithm is the first practical implementation of balanced fairness, just like the round-robin scheduling algorithm is a well-known practical implementation of the ideal processor-sharing (PS) service discipline.

Multi-server queues with specialized servers have already been considered in [1, 8, 25, 26] but these models assume that each job can be processed by only one server at a time. Our model is closer to the multi-server queue with redundant requests introduced by Gardner et al. [10, 11], where the class of a job defines the set of servers on which it is replicated. When several replicas of the same job are in service simultaneously on different servers, their service times are independent and the first instance to be completed stops the others. It is easy to see that, under the assumption of exponential service times, the two models are in fact equivalent. In both cases, the FCFS policy makes the system very sensitive to the job size distribution, so that the actual performance may vary significantly when the job sizes are not exponentially distributed with the same unit mean. Our objective in this paper is precisely to relax this assumption by designing a scheduling policy which makes the system insensitive to the job size distribution.

It turns out that our model belongs to the family of Order Independent (OI) queues [3, 16]. As observed in [16], OI queues generalize a number of queueing systems like BCMP networks under the FCFS or PS service discipline [2], multiserver stations with concurrent classes of customers (MSCCC) and multiserver stations with hierarchical concurrency constraints (MSHCC) [17, 18]. OI queues are known to be quasi-reversible [13]. In particular, the state of the queue has an explicit stationary distribution under the usual assumptions of Poisson arrivals and exponential service times. Moreover, the stationary distribution remains explicit in the presence of random routing, where jobs can leave or re-enter the queue upon service completion.

The first contribution of this paper is a scheduling algorithm which exploits this last property to mitigate the sensitivity to the job size distribution. Just like round-robin scheduling which implements the PS service discipline in the single-server case, our mechanism enforces insensitivity by interrupting the jobs frequently and moving them to the end of the queue. Routing is thus reinterpreted in terms of job interruptions and resumptions. The queue state is updated in the course of the job shiftings and the exponentially distributed sizes with unit mean in the multi-server queue now represent small fragments of the jobs. When the interruptions are frequent, each job tends to go back and forth in the queue and its average service rate is mainly determined by the

number of jobs of each class which are present at the same time.

This last observation motivates us to adopt a higher viewpoint. Specifically, we aggregate the state of the multi-server queue to only retain the number of jobs of each class, but not their arrival order. This aggregate state turns out to be an appropriate level of granularity to analyze the behavior of the queue. Its stationary measure is exactly that of a Whittle network [21] containing as many PS queues as there are classes in the original multi-server queue. This leads us to our second contribution: a new theoretical understanding of the multi-server queue. Using the state aggregation, we show in Theorem 1 that the queue is stable under any vector of acceptable arrival rates. In practice, it suggests that our algorithm will tend to stabilize the system whenever possible. Our second theoretical result, stated in Theorem 2, concerns the service rate received on average by each job as its position in the queue evolves. We show that the average per-class service rates when the number of jobs of each class is given are exactly those obtained by applying balanced fairness.

In addition to help us to understand the behavior of our algorithm, this equivalence with balanced fairness allows us to derive explicit expressions for the performance metrics of the multi-server queue with an arbitrary graph of compatibilities. Indeed, the insensitivity property satisfied by balanced fairness was used for instance in [6, 22, 23] to obtain simple and explicit recursion formulas for the performance metrics. Thanks to the aggregation we propose, these formulas can be applied as they are in the multi-server queue. They predict the exact performance of our algorithm when the job sizes are exponentially distributed. For an arbitrary job size distribution, we show by simulation that the system becomes approximately insensitive when the number of interruptions per job increases, so that the performance tends to that obtained under balanced fairness. We further observe that only a few interruptions per job actually suffice to reach approximate insensitivity.

The rest of the paper is organized as follows. In Section 2, we introduce the model and give the stability condition after recalling results on OI queues. In Section 3, it is shown that the resource allocation is balanced fairness in the presence of reentrant jobs. This result is used in Section 4 to design our scheduling algorithm. Some numerical results are presented in Section 5. Section 6 concludes the paper.

2. A multi-server queue

We consider a multi-server queue with N job classes and S servers. The class of a job may identify a client of the data center or a type of service; it defines the set of servers that can process this job. For each $i = 1, \dots, N$, class-i jobs enter the queue according to an independent Poisson process of intensity λ_i. The job sizes are independent, exponentially distributed with mean 1. We assume for now that each job leaves the queue immediately after service completion.

For each $i = 1, \dots, N$, we denote by $\mathcal{S}_i \subset \{1, \dots, S\}$ the set of servers that can process class-i jobs. Equivalently, these constraints can be represented as a bipartite graph of compatibilities between the N job classes and the S servers,

where there is an edge between class i and server s if and only if $s \in \mathcal{S}_i$. Each job can be served in parallel by multiple servers and each server processes the job sequentially in FCFS order. Hence, when there are several servers available for a job at its arrival, all these servers process this job. When the service of a job is complete, all the servers that were processing it are reallocated to the next job they can serve in the queue. There is no service preemption, so that at most one job of each class can be served at any given time.

We describe the evolution of the sequence of jobs in the queue, ordered by their arrival times. Thus the queue state is some sequence $c = (c_1, \ldots, c_n)$ of length n, where n is the number of jobs in the queue and $c_k \in \{1, \ldots, N\}$ is the class of job in position k, for each $k = 1, \ldots, n$, starting from the head of the queue. $\emptyset$ denotes the empty state, with $n = 0$.

When a job is in service on several servers, its service rate is the sum of the capacities of the servers that are processing it. Denoting by $\mu_s > 0$ the capacity of server s for each $s = 1, \ldots, S$, the total service rate in any state c is thus given by

$$\mu(c) = \sum_{s \in \bigcup_{k=1}^{n} \mathcal{S}_{c_k}} \mu_s.$$

For each $k = 1, \ldots, n$, the job in position k receives service at rate

$$\mu(c_1, \ldots, c_k) - \mu(c_1, \ldots, c_{k-1}) = \sum_{s \in \mathcal{S}_{c_k} \setminus \bigcup_{\ell=1}^{k-1} \mathcal{S}_{c_\ell}} \mu_s.$$

Observe that the total service rate in state c only depends on the set $A(c) = \{c_k : k = 1, \ldots, n\}$ of active classes in state c. Hence, for each $A \subset \{1, \ldots, N\}$, we can denote by $\mu(A)$ the service rate in any state c whose set of active classes is A. This is a submodular function, as a weighted cover set function [9, 20].

Order Independent queues. This multi-server queue turns out to be a special case of Order Independent (OI) queues. These were introduced by Berezner and Krzesinski [3, 16] as a new class of multi-class quasi-reversible queues. The description of an OI queue is the same as for the multi-server queue except that the total service rate μ can be any function of the queue state c which satisfies the following properties:

- Monotonicity: $\mu(c_1, \ldots, c_n) \leq \mu(c_1, \ldots, c_n, i)$ for any state c and class i,
- Order-independence: $\mu(c_1, \ldots, c_n) = \mu\left(c_{\sigma(1)}, \ldots, c_{\sigma(n)}\right)$ for any state c and permutation σ of $1, \ldots, n$.

Additionally, it is assumed that $\mu(\emptyset) = 0$ and $\mu(c) > 0$ for all $c \neq \emptyset$. The total service rate $\mu(c)$ is allocated to jobs in the order of their arrival in the sense that the job in position k receives service at rate $\mu(c_1, \ldots, c_k) - \mu(c_1, \ldots, c_{k-1})$. In particular, the service received by a job does not depend on the jobs arrived later in the queue. One can easily verify that the service rate of our multi-server queue satisfies these properties.

Stationary measure. The queue state c defines a Markov process on $\{1,\dots,N\}^*$. Since the multi-server queue is a special case of OI queues, it follows from [16, Theorem 2.2] that this queue is quasi-reversible, with stationary measure

$$\forall c \in \{1,\dots,N\}^*, \quad \pi(c) = \pi(\emptyset) \prod_{k=1}^{n} \frac{\lambda_{c_k}}{\mu(A(c_1,\dots,c_k))}. \tag{1}$$

This formula was also derived in [11, Theorem 1] for multi-server queues with redundant requests. However, the observation that the multi-server queue is quasi-reversible is critical because it allows us to add random routing between job classes [13]. As we will see in Sections 3 and 4, this result plays a key role in the design of our algorithm.

Aggregate state. As in [16], we consider the number of jobs of each class in the queue, independently of their arrival order. We denote by $x = (x_1,\dots,x_N)$ the corresponding aggregate state, where x_i is the number of class-i jobs in the queue. This defines a stochastic process on $\mathbb{N}^N$, which is not a Markov process in general. We refer to the stationary measure of the aggregate state x as

$$\bar{\pi}(x) = \sum_{c:|c|=x} \pi(c), \tag{2}$$

where $|c| \in \mathbb{N}^N$ denotes the vector of the numbers of jobs of each class in state c. We also denote the set of active classes in any state x by $A(x) = \{i : x_i > 0\}$.

It was proved in [16] that the stationary measure of the aggregate state is given by

$$\bar{\pi}(x) = \bar{\pi}(0)\Phi(x) \prod_{i=1}^{N} \lambda_i^{x_i}, \tag{3}$$

where the function Φ satisfies the recursion $\Phi(0) = 1$ and, for each $x \neq 0$,

$$\Phi(x) = \frac{1}{\mu(A(x))} \sum_{i \in A(x)} \Phi(x - e_i), \tag{4}$$

e_i being the N-dimensional vector with 1 in component i and 0 elsewhere, for any $i = 1,\dots,N$.

Stability condition. The following key result is proved in the appendix.

Theorem 1. *The multi-server queue is stable, in the sense that the underlying Markov process is ergodic, if and only if*

$$\forall A \subset \{1,\dots,N\}, A \neq \emptyset, \quad \sum_{i \in A} \lambda_i < \mu(A). \tag{5}$$

In the rest of the paper, we assume that this condition is satisfied and we denote by π the stationary distribution of the queue state.

3. Average resource allocation

Re-entrant jobs. Since the multi-server queue is quasi-reversible, the stationary distribution of the queue state is not modified by the addition of routing between classes as long as the effective arrival rates remain constant [13]. Assume for instance that each job leaves the queue with probability p and re-enters as a job of the same class with probability $1-p$, for some $p \in (0,1]$. The external arrival rate of class-i jobs is taken equal to $\lambda_i p$ so that the effective arrival rate of class-i jobs remains equal to λ_i. The stationary distribution of the queue state c is still given by (1), independently of p. Each job re-enters the queue $1/p$ times on average, which tends to infinity when $p \to 0$.

In the limit, it is not relevant to consider the instantaneous service rate of each job depending on its position in the queue; the metric of importance is the service rate received *on average* by each job when the number of jobs of each class in the queue is given, corresponding to the aggregate state x. The objective of this section is precisely to gain insights into the steady-state behavior of the multi-server queue viewed through its aggregate state.

Whittle network. As we will see in Theorem 2 below, the stationary distribution (3) of the aggregate state x of the multi-server queue is that of the state of a Whittle network [21] of N queues.

A Whittle network of N queues is a network of N processor-sharing queues with state-dependent service rates. The network state is described by the vector $x = (x_1, \ldots, x_N)$ giving the number of jobs at each queue. The key feature of a Whittle network is that the relative variations of the service rates $\phi_1, \ldots, \phi_N$ of the queues are constrained by the following balance property:

$$\forall x \in \mathbb{N}^N, \quad \forall i,j : x_i > 0, x_j > 0, \quad \phi_i(x)\phi_j(x-e_i) = \phi_i(x-e_j)\phi_j(x). \tag{6}$$

This balance property is equivalent to the insensitivity property, i.e., the fact that the stationary distribution of the network state is independent of the job size distribution beyond the mean [4].

The service rates $\phi_1, \ldots, \phi_N$ satisfy the balance property (6) if and only if there is a balance function Φ such that $\Phi(0) = 1$ and

$$\forall x \in \mathbb{N}^N, \quad \forall i = 1, \ldots, N, \quad \phi_i(x) = \begin{cases} \frac{\Phi(x-e_i)}{\Phi(x)} & \text{if } x_i > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

From this it is easy to show that the steady-state distribution $\bar{\pi}$ given by (3) with the function Φ given by (7) satisfies the local balance equations of the network.

Conversely, the balance function Φ uniquely defines the service rates of the queues of a Whittle network through (7). In particular, there exists a unique Whittle network of N queues with per-queue arrival rates $\lambda_1, \ldots, \lambda_N$ whose balance function is given by (4). The stationary distribution of this network state is exactly the stationary distribution (3) of the aggregate state of the multi-server queue.

The following two key results specify the relation between the average per-class service rates in the multi-server queue and the service rates of the queues in this equivalent Whittle network.

Theorem 2. *The stationary distribution of the aggregate state of the multi-server queue is that of the state of a Whittle network of N queues, with arrival rates $\lambda_1, \ldots, \lambda_N$ and state-dependent service rates $\phi_1, \ldots, \phi_N$ given by*

$$\phi_i(x) = \sum_{c:|c|=x} \frac{\pi(c)}{\bar{\pi}(x)} \mu_i(c), \tag{8}$$

where $\mu_i(c)$ is the service rate of the first class-i job in state c of the multi-server queue, for each $c \in \{1, \ldots, N\}^*$ and $i = 1, \ldots, N$.

Proof. As observed earlier, the stationary distribution (3) is exactly the stationary distribution of the state x of a Whittle network of N queues with arrival rates $\lambda_1, \ldots, \lambda_N$ and service rates $\phi_1, \ldots, \phi_N$ given by (6), where Φ is the balance function given by (4). We just need to verify that these service rates satisfy (8).

Let $x \in \mathbb{N}^N$ and $i = 1, \ldots, N$ such that $x_i > 0$. We have

$$\phi_i(x) = \frac{\Phi(x - e_i)}{\Phi(x)} = \frac{\bar{\pi}(x - e_i)\lambda_i}{\bar{\pi}(x)} = \frac{1}{\bar{\pi}(x)} \sum_{c:|c|=x-e_i} \pi(c)\lambda_i.$$

The quasi-reversibility of the multi-server queue ensures that the following partial balance equation is satisfied in any state c (see the proof of [16, Theorem 2.2] for more details):

$$\begin{aligned}\pi(c)\lambda_i = \sum_{k=1}^{n+1} &\pi(c_1, \ldots, c_{k-1}, i, c_k, \ldots, c_n) \\ &\times \left(\mu(A(c_1, \ldots, c_{k-1}, i)) - \mu(A(c_1, \ldots, c_{k-1}))\right).\end{aligned}$$

Letting $n = x_1 + \ldots + x_N$, we deduce that

$$\begin{aligned}\sum_{c:|c|=x-e_i} \pi(c)\lambda_i &= \sum_{c:|c|=x-e_i} \sum_{k=1}^{n} \pi(c_1, \ldots, c_{k-1}, i, c_k, \ldots, c_{n-1}) \\ &\qquad \times \left(\mu(A(c_1, \ldots, c_{k-1}, i)) - \mu(A(c_1, \ldots, c_{k-1}))\right), \\ &= \sum_{c:|c|=x} \pi(c) \sum_{\substack{k=1 \\ c_k=i}}^{n} \left(\mu(A(c_1, \ldots, c_{k-1}, c_k)) - \mu(A(c_1, \ldots, c_{k-1}))\right), \\ &= \sum_{c:|c|=x} \pi(c)\mu_i(c).\end{aligned}$$

This equation remains valid for any state x and class i such that $x_i = 0$. □

Corollary 1. *For each $x \in \mathbb{N}^N$, the vector of service rates $\phi(x) = (\phi_1(x), \ldots, \phi_N(x))$ belongs to the capacity set*

$$\mathcal{C} = \left\{ \phi \in \mathbb{R}_+^N : \ \forall A \subset \{1, \ldots, N\}, \ \sum_{i \in A} \phi_i \leq \mu(A) \right\}$$

and satisfies

$$\sum_{i \in A(x)} \phi_i(x) = \mu(A(x)).$$

Proof. Let $x \in \mathbb{N}^N$. For all $A \subset \{1, \ldots, N\}$, we have by (8),

$$\sum_{i \in A} \phi_i(x) = \sum_{c:|c|=x} \frac{\pi(c)}{\bar{\pi}(x)} \sum_{i \in A} \mu_i(c) \leq \sum_{c:|c|=x} \frac{\pi(c)}{\bar{\pi}(x)} \mu(A) = \mu(A).$$

For $A = A(x)$, we have

$$\sum_{i \in A(x)} \mu_i(c) = \mu(A(x))$$

for each c such that $|c| = x$, so that the above inequality is an equality. □

Balanced fairness. By Theorem 2, the average service rates in the multi-server queue satisfy the balance property (6). In view of Corollary 1, the resource allocation is also Pareto-efficient in the sense that the server resources are always maximally consumed. The unique resource allocation which satisfies these two properties is known as balanced fairness [5].

Going back to the motivating example with re-entrant jobs, in the limit where $p \to 0$, the external arrivals and departures become rare and the jobs tend to re-enter the queue several times. The detailed queue state evolves with these frequent job shifts, while the aggregate state remains constant. On average, all jobs of class i tend to be served at the same service rate in aggregate state x, with total service rate $\phi_i(x)$. When the queue contains only one server, it means that the capacity of this server is divided equally among all jobs in the queue, similarly to round-robin scheduling. In general, this corresponds to the above Whittle network where each of the N queues applies the processor-sharing service discipline. Such a queueing system is known to have the insensitivity property described above. This property will be exploited in the next section to design a scheduling algorithm in computer clusters based on re-entrant jobs after forced service interruptions.

Performance metrics. Several works have focused on predicting the performance of systems under balanced fairness, see for instance [6, 22, 23]. Their results can be reused as they are to predict the performance of the multi-server queue. Indeed, the above aggregation results show that any performance metric which can be expressed in terms of the aggregate state in the multi-server queue is actually equal to the corresponding metric in the equivalent Whittle network. This is stated more formally in the following corollary, which follows from Theorem 2.

Corollary 2. *Consider a function f defined on $\{1,\dots,N\}^*$ which is order-independent, in the sense that there exists a function g defined on $\mathbb{N}^N$ such that $f(c) = g(|c|)$ for all $c \in \{1,\dots,N\}^*$. Then the expected value of f applied to the state c of the multi-server queue is equal to the expected value of g applied to the state x of the equivalent Whittle network.*

Proof. Gathering the detailed queue states which correspond to the same aggregate state, we obtain directly

$$\begin{aligned}\sum_{c\in\{1,\dots,N\}^*} \pi(c)f(c) &= \sum_{c\in\{1,\dots,N\}^*} \pi(c)g(|c|),\\ &= \sum_{x\in\mathbb{N}^N} \left(\sum_{c:|c|=x} \pi(c)\right) g(x),\\ &= \sum_{x\in\mathbb{N}^N} \bar{\pi}(x)g(x).\end{aligned}$$

□

Note that this result holds for any stationary measure π. In particular, taking the measure π such that $\pi(\emptyset) = 1$ in the multi-server queue (that is, $\bar{\pi}(0) = 1$ in the equivalent Whittle network) and for f the constant function equal to 1, we obtain that the normalization constants in the multi-server queue and in the equivalent Whittle network are equal.

A metric of importance is the mean number of jobs of a given class in the multi-server queue, from which we can deduce the mean delay (or equivalently the mean service rate) perceived by the jobs of class i, for each $i = 1,\dots,N$. Coming back to the stationary distribution π with the function f which counts the number of jobs of a given class in the multi-server queue, we deduce from Corollary 2 that the mean number of class-i jobs in the multi-server queue is equal to the mean number of jobs at queue i in the equivalent Whittle network, for each $i = 1,\dots,N$.

Hence, the recursive formulas of [22, Theorem 4] and [23, Theorem 1] give directly the normalization constant of the stationary distribution, as well as the mean number of jobs of each class in the multi-server queue. The numerical results presented in Section 5 are based on this observation.

4. A scheduling algorithm for computer clusters

We apply the previous results to the problem of resource sharing in computer clusters. Consider a cluster of S servers. For all $s = 1,\dots,S$, we denote by C_s the service capacity of server s, in floating-point operations per second (flops). Any incoming job consists of some random number of floating-point operations, referred to as the job size, and is assigned some set of servers. This assignment, possibly random, may depend on the type of the job but not on the system state (e.g., the number of ongoing jobs). It is fixed for the entire life of the job

in the system. The job can then be processed in parallel by any subset $\mathcal{S}$ of the servers in its assignment, at rate $\sum_{s\in\mathcal{S}} C_s$. Job sizes are assumed i.i.d. with mean σ.

Balanced fairness. We aim at sharing the service capacity of the cluster according to balanced fairness, so that the stationary distribution of the number of jobs of each class is independent of the job size distribution beyond the mean [5]. Applying the FCFS service discipline to each server is clearly not suitable. For $S = 1$ for instance, the system reduces to a single-server FCFS queue, which is known to be very sensitive to the job size distribution. For $S \geq 1$, the system corresponds to the multi-server queue described in Section 2, with service rates $\mu_s = C_s/\sigma$ for all $s = 1, \ldots, S$, provided job sizes are i.i.d. exponential with mean σ.

We apply the idea of re-entrant jobs mentioned in Section 3. Specifically, we interrupt each service after some exponential time and force the corresponding job to re-enter the queue as a new job of the same type, with the same server assignment, so that the service can be resumed later and the resources can be reallocated. Observe that, when job sizes are i.i.d. exponential with mean σ, the stationary distribution of the aggregate state remains unchanged by the quasi-reversibility of the OI queue. When the frequency of service interruptions increases, the resources tend to be shared fairly, in the sense of balanced fairness, and the stationary distribution becomes insensitive to the job size distribution beyond the mean. For $S = 1$ for instance, the system tends to a single-server PS queue, which is known to have the insensitivity property. For $S \geq 1$, the system tends to a Whittle network of PS queues, which is also known to have the insensitivity property [4].

Scheduling algorithm. A single virtual queue is used to allocate servers to jobs. Any incoming job is put at the end of the queue. Each server s interrupts the job in service, if any, after some exponential time with parameter C_s/θ, for some $\theta > 0$. Observe that θ can be interpreted as the mean number of floating-point operations before service interruption. Any interrupted job releases *all* servers that process this job and is moved to the end of the queue as a new job. The released resources are reallocated according to the same service discipline, accounting for the new order in the queue. Note that the interrupted service may be resumed immediately or later, when some resources become available, depending on the state of the queue.

The pseudo-code of the algorithm is given in Algorithm 1, where $a_s \in \{0, 1, \ldots, N\}$ denotes the activity state of server s ($a_s = 0$ if server s is idle and $a_s = i$ if server s is processing a job of class i) and $t_s \in \{\text{on}, \text{off}\}$ indicates the state of the timer that triggers service interruption at server s (when set on, the timer has an exponential distribution with parameter C_s/θ). The algorithm depends on a single parameter θ, which determines the mean number of service interruptions per job. This should be compared to the mean job size σ. Specifically, the ratio $m = \sigma/\theta$ corresponds to the mean number of service interruptions per job. When $m \to \infty$, services are frequently interrupted and

```
on job arrival
begin
    i ← job class
    enqueue job
    n ← n + 1
    for all s ∈ S_i do
        if a_s = 0 then
            a_s ← i
            t_s ← on
        end
    end
end

on job departure
begin
    i ← job class
    k ← job position in the queue
    dequeue job
    n ← n − 1
    for all s ∈ S_i do
        if a_s = i then
            a_s ← 0
            t_s ← off
        end
    end
    while k ≤ n do
        for all s ∈ S_{c_k} ∩ S_i do
            if a_s = 0 then
                a_s ← c_k
                t_s ← on
            end
        end
        k ← k + 1
    end
end
```

```
on timer expiration
begin
    s ← server
    i ← a_s
    k ← job position in the queue
    interrupt job service
    move job to the end of the queue
    for all s ∈ S_i do
        if a_s = i then
            a_s ← 0
            t_s ← off
        end
    end
    while k ≤ n do
        for all s ∈ S_{c_k} ∩ S_i do
            if a_s = 0 then
                a_s ← c_k
                t_s ← on
            end
        end
        k ← k + 1
    end
end
```

Algorithm 1: Scheduling algorithm based on random service interruptions.

the corresponding resource allocation tends to balanced fairness, as mentioned in Section 3, an allocation that has the insensitivity property; when $m \to 0$, services are almost never interrupted and the service discipline is approximately FCFS per server, which is highly sensitive to the job size distribution. We shall see in the following section that, for large systems with random assignment, setting $m = 1$ is in fact sufficient to get approximate insensitivity, i.e., it is sufficient in practice to interrupt each job only *once* on average.

5. Numerical results

In this section, we provide numerical results showing the performance of the algorithm described above. We are specifically interested in evaluating the mean number m of interruptions per job which is sufficient in practice to obtain approximate insensitivity to the job size distribution.

Job size distribution. As in Section 4, the job sizes are assumed i.i.d. To test the sensitivity, we successively evaluate the performance of our algorithm under three job size distributions.

We first consider job sizes with a bimodal number of exponentially distributed phases. More precisely, the size of any incoming job is a sum of independent random variables which are exponentially distributed with mean ς. The number of these random variables follows a bimodal distribution: it is equal to n_1 with probability p_1 and to n_2 with probability p_2, for some p_1, p_2 such that $p_1 + p_2 = 1$. We let $\varsigma = 1/5, n_1 = 25$, $n_2 = 1$, $p_1 = 1/6$ and $p_2 = 5/6$. The mean job size is given by $\sigma = (p_1 n_1 + p_2 n_2)\varsigma = 1$ while the standard deviation is approximately equal to 1.84.

We consider a second alternative where the job size distribution is hyperexponential: any incoming job has an exponential distribution with mean σ_1 with probability p_1 and an exponential distribution with mean σ_2 with probability p_2, for some p_1, p_2 such that $p_1 + p_2 = 1$. We let $\sigma_1 = 5$, $\sigma_2 = 1/5$, $p_1 = 1/6$, $p_2 = 5/6$, corresponding to a mean job size $\sigma = p_1\sigma_1 + p_2\sigma_2 = 1$ and standard deviation approximately equal to 2.05.

Finally, we consider job sizes with a heavy-tailed number of exponential phases. Like for the bimodal case, the size of any incoming job is a sum of independent random variables which are exponentially distributed with mean ς. The number of these random variables follows a Zipf distribution with parameters $K \in \mathbb{N}$ and $\alpha > 0$: for each $k = 1, \ldots, K$, the probability that there are k terms in the sum is proportional to $1/k^\alpha$. We let $\varsigma = 1$, $K = 200$ and $\alpha = 2$. The mean job size is then given by

$$\sigma = \frac{\sum_{k=1}^{K} \frac{1}{k^{\alpha-1}}}{\sum_{k=1}^{K} \frac{1}{k^\alpha}}\varsigma,$$

approximately equal to 3.58, while the standard deviation is approximately equal to 10.61.

Performance metrics. We measure the performance in terms of mean service rate and mean delay. Let $\phi_i(x)$ be the service rate of class-i jobs in state x, as defined by (8). The mean service rate of any class-i job is then given by:

$$\gamma_i = \frac{\sum_x \bar{\pi}(x)\phi_i(x)}{\sum_x \bar{\pi}(x)x_i}.$$

By conservation, we have:

$$\gamma_i = \frac{\lambda_i \sigma}{\sum_x \bar{\pi}(x)x_i}.$$

Observe that γ_i cannot exceed the maximum service rate of class-i jobs, given by $\sum_{s\in\mathcal{S}_i} C_s$. The mean delay δ_i of any class-i job follows from the mean number of class-i jobs by Little's law, and is inversely proportional to the mean service rate:

$$\delta_i = \frac{\sum_x \bar{\pi}(x)x_i}{\lambda_i} = \frac{\sigma}{\gamma_i}. \tag{9}$$

Performance evaluation. There are S servers and N job classes. Class-i jobs arrive according to a Poisson process with intensity λ_i. The mean number of interruptions per job is given by $m = \sigma/\theta$, where σ is the mean job size and θ is the parameter of the algorithm used to set the random timers. We compare the results for $m = 1$ and $m = 5$ with those obtained under FCFS policy (that is, without service interruption) and balanced fairness.

The performance metrics under balanced fairness will be given in closed form for the configurations considered below. They give the performance of our algorithm and of FCFS policy when the job size distribution is exponential. We resort to simulations to assess the performance under the three job size distributions listed earlier. Each simulation point follows from the average of 100 independent runs, each corresponding to 10^6 jumps of the corresponding Markov process, after a warm-up period of 10^6 points; the corresponding 95% confidence intervals are drawn in semitransparent on the figures.

Three servers. We first consider a toy example with $S = 3$ servers and $N = 2$ job classes. Servers 1 and 2 are dedicated to job classes 1 and 2, respectively, while server 3 is shared by all jobs. In view of Theorem 1, the stability condition is:

$$\lambda_1 < \mu_1 + \mu_3, \quad \lambda_2 < \mu_2 + \mu_3, \quad \lambda_1 + \lambda_2 < \mu_1 + \mu_2 + \mu_3.$$

Define the corresponding loads:

$$\rho_1 = \frac{\lambda_1}{\mu_1 + \mu_3}, \quad \rho_2 = \frac{\lambda_2}{\mu_2 + \mu_3}, \quad \rho = \frac{\lambda_1 + \lambda_2}{\mu_1 + \mu_2 + \mu_3}.$$

Observing that the capacity set is that of a tree network [6], we deduce the mean service rates under balanced fairness:

$$\gamma_1 = \left(\frac{1}{\mu(1-\rho)} + \frac{\frac{\mu_2}{\mu_1+\mu_3}\frac{1-\rho_2}{1-\rho_1}}{\mu - (\mu_1+\mu_3)\rho_1 - (\mu_2+\mu_3)\rho_2 + \mu_3\rho_1\rho_2} \right)^{-1}, \tag{10}$$

$$\gamma_2 = \left(\frac{1}{\mu(1-\rho)} + \frac{\frac{\mu_1}{\mu_2+\mu_3}\frac{1-\rho_1}{1-\rho_2}}{\mu - (\mu_1+\mu_3)\rho_1 - (\mu_2+\mu_3)\rho_2 + \mu_3\rho_1\rho_2} \right)^{-1} \tag{11}$$

with $\mu = \mu_1 + \mu_2 + \mu_3$. The mean delays follow by (9). Explicit formulas for the performance metrics under this assignment graph were also derived in [11] in the context of multi-server queues with redundant requests. Recall that these are the exact performance metrics when the job size distribution is exponential. The results are shown in Figures 1 and 2 with respect to the load ρ, for $\lambda_1 = \lambda_2$. In Figure 1, the system is symmetric and the maximum service rate is 2 for both classes. In Figure 2, the system is asymmetric: class-1 jobs (in blue) can be served by servers 1,3 and thus have a maximum service rate of 2; class-2 jobs (in red) can be served by server 3 only and thus have a maximum service rate of 1.

Applying our scheduling algorithm with only $m = 1$ (that is, 1 service interruption per job on average) brings a significant improvement compared to FCFS policy. For $m = 5$, performance is very close to that of balanced fairness and approximately insensitive (i.e., very close to that obtained for an exponential job size distribution) even for job sizes with a Zipf number of exponential phases.

Large system with random assignment. We now consider a large system of $S = 100$ servers, each with unit service rate. Each incoming job is assigned d servers chosen uniformly at random, corresponding to $N = \binom{S}{d}$ job classes, as considered in [10]. The mean service rate and the mean delay follow from an explicit formula for the mean number of jobs in the queue derived in [10]. The simulation results are obtained in the conditions described above. The results for $d = 2$ and $d = 3$ are shown in Figures 3 and 4, respectively. We observe that performance is very close to that of balanced fairness, even for low values of m. It is sufficient in practice to set $m = 1$, corresponding to only *one* service interruption per job on average.

6. Conclusion

We have introduced a new scheduling algorithm to allocate the resources of a computer cluster according to balanced fairness. This algorithm, which is based on service interruptions and resumptions, can be viewed as an extension of round-robin scheduling algorithm in the context of resource pooling. Its performance was studied by considering a new queueing model where jobs can be processed in parallel by several servers. We have observed in particular that the aggregate state of the queue is that of a Whittle network, and deduced the

insensitivity property in the limit of an infinite number of service interruptions per job. This has in turn allowed us to derive explicit expressions for the performance metrics with an arbitrary graph of compatibilities. The performance of the system when the number of service interruptions per job is finite was assessed by simulation. We observed that only a few interruptions per job are sufficient in practice to obtain approximate insensitivity.

Our objectives for the future work are twofold. First, we aim at refining our understanding of the system presented in this paper. This notably involves assessing analytically the impact of the mean number of service interruptions on the sensitivity of the resulting resource allocation. We would also like to perform more simulations to compare the performance of our algorithm with that of other existing scheduling policies, regarding both the insensitivity to the job size distribution and the efficiency of the resource utilization.

A second step would be to extend the current model and algorithm and include practical constraints which are inherent to parallel computing. We can notably mention the cost of coordination between servers, not only during the service but also upon service interruption. Besides it would be interesting to consider alternative ways of enforcing frequent service interruptions, which do not rely on exponentially distributed timers but instead utilize the structure of the real system considered. Depending on the application, it may be possible for instance to pre-cut the jobs into smaller tasks of comparable size. Finally, we would like to explore other variants of the queueing model, including the representation of fork-join tasks using stochastic Petri networks [19], the presence of negative customers [12, 24] and batch services [7] and the case of loss networks [15].

References

[1] I. Adan, M. Boon, A. Bušić, J. Mairesse, and G. Weiss. Queues with skill based parallel servers and a FCFS infinite matching model. *SIGMETRICS Perform. Eval. Rev.*, 41(3):22–24, Jan. 2014.

[2] F. Baskett, K. Chandy, R. Muntz, and F. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2):248–260, Apr. 1975.

[3] S. A. Berezner and A. E. Krzesinski. Order independent loss queues. *Queueing Systems*, 23(1):331–335, 1996.

[4] T. Bonald and A. Proutière. Insensitivity in processor-sharing networks. *Performance Evaluation*, 2002.

[5] T. Bonald and A. Proutière. Insensitive bandwidth sharing in data networks. *Queueing Syst.*, 44(1):69–100, 2003.

[6] T. Bonald and J. Virtamo. Calculating the flow level performance of balanced fairness in tree networks. *Performance Evaluation*, 2004.

[7] R. Boucherie and N. van Dijk. Product forms for queueing networks with state-dependent multiple job transitions. *Advances in Applied Probability*, pages 152–187, 1991.

[8] R. Caldentey, E. H. Kaplan, and G. Weiss. FCFS infinite bipartite matching of servers and customers. *Advances in Applied Probability*, 41(3):695–730, 009 2009.

[9] J. Edmonds. Submodular functions, matroids, and certain polyhedra. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization — Eureka, You Shrink!: Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers*, pages 11–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[10] K. Gardner, M. Harchol-Balter, A. Scheller-Wolf, M. Velednitsky, and S. Zbarsky. Redundancy-d: The power of d choices for redundancy. *Operations Research*, 2016.

[11] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytia. Reducing latency via redundant requests: Exact analysis. In *Proceedings of ACM SIGMETRICS 2015*, pages 347–360, New York, NY, USA, 2015. ACM.

[12] E. Gelenbe. G-networks with signals and batch removal. *Probability in the Engineering and Informational Sciences*, 7(03):335–342, 1993.

[13] F. Kelly. *Reversibility and Stochastic Networks*. Wiley, Chichester, 1979.

[14] F. Kelly, L. Massoulié, and N. Walton. Resource pooling in congested networks: proportional fairness and product form. *Queueing Systems*, 63(1-4):165–194, 2009.

[15] F. P. Kelly. Loss networks. *Ann. Appl. Probab.*, 1(3):319–378, 08 1991.

[16] A. E. Krzesinski. Order independent queues. In R. J. Boucherie and N. M. van Dijk, editors, *Queueing Networks: A Fundamental Approach*, pages 85–120. Springer US, Boston, MA, 2011.

[17] A. E. Krzesinski and R. Schassberger. Product Form Solutions for Multiserver Centers with Hierarchical Concurrency Constraints. *Probability in the Engineering and Informational Sciences*, 6(2):147–156, Apr. 1992.

[18] J.-Y. Le Boudec. A BCMP Extension to Multiserver Stations with Concurrent Classes of Customers. In *Proceedings of the 1986 ACM SIGMETRICS Joint International Conference on Computer Performance Modelling, Measurement and Evaluation*, SIGMETRICS '86/PERFORMANCE '86, pages 78–91, New York, NY, USA, 1986. ACM.

[19] A. Marin, S. Balsamo, and P. G. Harrison. Analysis of stochastic petri nets with signals. *Performance Evaluation*, 69(11):551 – 572, 2012.

[20] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[21] R. Serfozo. *Introduction to Stochastic Networks*. Stochastic Modelling and Applied Probability. Springer New York, 1999.

[22] V. Shah and G. de Veciana. High-performance centralized content delivery infrastructure: Models and asymptotics. *IEEE/ACM Transactions on Networking*, 23(5):1674–1687, Oct 2015.

[23] V. Shah and G. de Veciana. Impact of fairness and heterogeneity on delays in large-scale centralized content delivery systems. *Queueing Systems*, 83(3):361–397, 2016.

[24] T. H. D. Thi, J. M. Fourneau, and M. A. Tran. Networks of order independent queues with signals. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 131–140, Aug 2013.

[25] J. N. Tsitsiklis and K. Xu. Flexible queueing architectures. *arXiv preprint arXiv:1505.07648*, 2015.

[26] J. Visschers, I. Adan, and G. Weiss. A product form solution to a system with multi-type jobs and multi-type servers. *Queueing Systems*, 70(3):269–298, 2012.

[27] N. S. Walton et al. Flow-level convergence and insensitivity for multi-class queueing networks. *Stochastic Systems*, 2(1):115–148, 2012.

Appendix

We prove Theorem 1 which states that the multi-server queue is stable if and only if

$$\forall A \subset \{1,\dots,N\}, A \neq \emptyset, \quad \sum_{i\in A} \lambda_i < \mu(A). \tag{5}$$

Necessary condition. Assume that $\mu(A) \leq \sum_{i\in A} \lambda_i$ for some non-empty set $A \subset \{1,\dots,N\}$. For any $x \in \mathbb{N}^N \setminus \{0\}$ such that $A(x) \subset A$, we also have $\mu(A(x)) \leq \mu(A)$ since μ is non-decreasing, so that

$$\Phi(x) = \frac{1}{\mu(A(x))} \sum_{i\in A(x)} \Phi(x-e_i) \geq \frac{1}{\sum_{i\in A}\lambda_i} \sum_{i\in A(x)} \Phi(x-e_i),$$

and by induction,

$$\Phi(x) \geq \frac{n!}{\prod_{i\in A} x_i!} \left(\frac{1}{\sum_{i\in A}\lambda_i}\right)^n,$$

where $n = \sum_{i\in A} x_i$. Hence we obtain

$$\begin{aligned}\sum_{x\in\mathbb{N}^N} \Phi(x)\prod_{i=1}^N {\lambda_i}^{x_i} &\geq \sum_{\substack{x\in\mathbb{N}^N:\\ A(x)\subset A}} \Phi(x)\prod_{i=1}^N {\lambda_i}^{x_i} = \sum_{n\geq 0}\sum_{\substack{x\in\mathbb{N}^N:\\ A(x)\subset A,\\ \sum_{i\in A} x_i = n}} \Phi(x)\prod_{i=1}^N {\lambda_i}^{x_i},\\ &\geq \sum_{n\geq 0}\sum_{\substack{x\in\mathbb{N}^N:\\ A(x)\subset A,\\ \sum_{i\in A} x_i = n}} \frac{n!}{\prod_{i\in A} x_i!}\prod_{i\in A}\left(\frac{\lambda_i}{\sum_{i\in A}\lambda_i}\right)^{x_i},\\ &= +\infty.\end{aligned}$$

Sufficient condition. We first prove the following lemma.

Lemma 1. *Let Ψ be such that $\Psi(0) = 1$ and for all $x \neq 0$,*

$$\sum_{i\in A(x)} \frac{\Psi(x-e_i)}{\Psi(x)} \leq \mu(A(x)). \tag{A.1}$$

Then $\Phi(x) \leq \Psi(x)$ for all $x \in \mathbb{N}^N$.

Proof. The proof is by induction on $n = \sum_{i=1}^N x_i$. The condition is true for $n = 0$ since $\Phi(0) = \Psi(0) = 1$. Now let $n \geq 1$ and assume that $\Phi(x) \leq \Psi(x)$ for all $x \in \mathbb{N}^N$ with $\sum_{i=1}^N x_i \leq n-1$. For each $x \in \mathbb{N}^N$ with $\sum_{i=1}^N x_i = n$, we obtain

$$\Psi(x) \geq \frac{\sum_{i\in A(x)} \Psi(x-e_i)}{\mu(A(x))} \geq \frac{\sum_{i\in A(x)} \Phi(x-e_i)}{\mu(A(x))} = \Phi(x),$$

where the first inequality holds because Ψ satisfies (A.1) and the second holds by the induction assumption. □

Assume that the stability condition (5) is satisfied. The proof consists in choosing a function Ψ that satisfies the assumptions of Lemma 1 and such that

$$\sum_{x\in\mathbb{N}^N} \Psi(x)\prod_{i=1}^N {\lambda_i}^{x_i} < +\infty.$$

In view of (5), there exists $\eta \in \mathbb{R}_+^N$ such that

$$\forall i = 1,\dots,N, \quad \lambda_i < \eta_i \quad \text{and} \quad \forall A \subset \{1,\dots,N\}, \quad \sum_{i\in A} \eta_i < \mu(A).$$

We can choose for instance

$$\forall i = 1,\dots,N, \quad \eta_i = \lambda_i + \frac{1}{2}\min_{A\subset\{1,\dots,N\}: i\in A}\left(\frac{\mu(A) - \sum_{j\in A}\lambda_j}{|A|}\right).$$

Now consider the balance function Ψ defined by

$$\forall x \in \mathbb{N}^N, \quad \Psi(x) = \prod_{i=1}^{N} \frac{1}{{\eta_i}^{x_i}}.$$

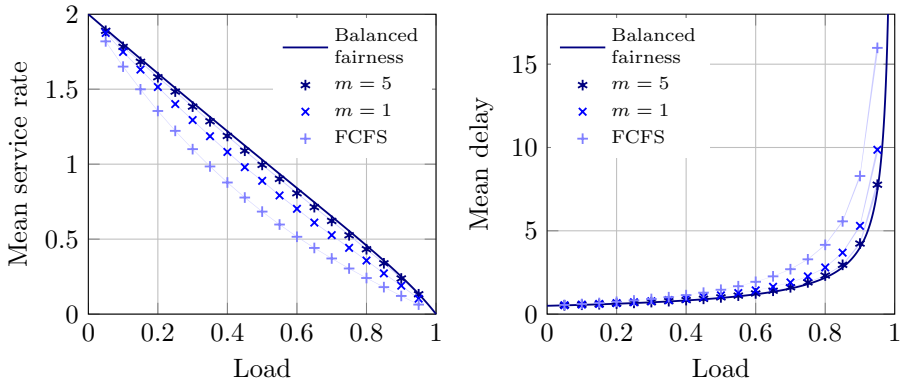
We have $\Psi(0) = 1$ and, for each $x \in \mathbb{N}^N \setminus \{0\}$,

$$\sum_{i \in A(x)} \frac{\Psi(x - e_i)}{\Psi(x)} = \sum_{i \in A(x)} \eta_i < \mu(A).$$

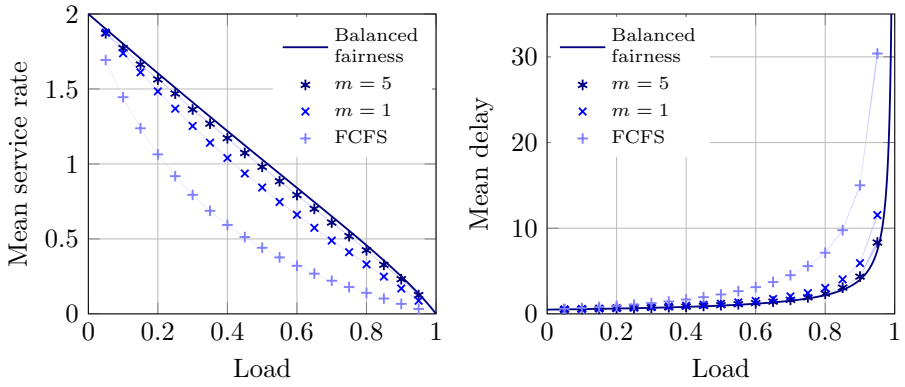
We can thus apply Lemma 1 to Ψ and we deduce that $\Phi(x) \leq \Psi(x)$ for all $x \in \mathbb{N}^N$. It follows that

$$\sum_{x \in \mathbb{N}^N} \Phi(x) \prod_{i=1}^{N} {\lambda_i}^{x_i} \leq \sum_{x \in \mathbb{N}^N} \Psi(x) \prod_{i=1}^{N} {\lambda_i}^{x_i} = \sum_{x \in \mathbb{N}^N} \prod_{i=1}^{N} \left(\frac{\lambda_i}{\eta_i} \right)^{x_i} < +\infty,$$

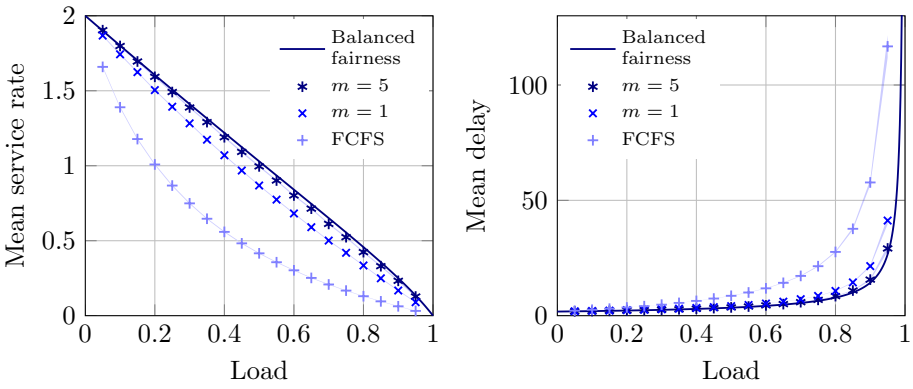
which concludes the proof.



(a) Bimodal number of exponentially distributed phases

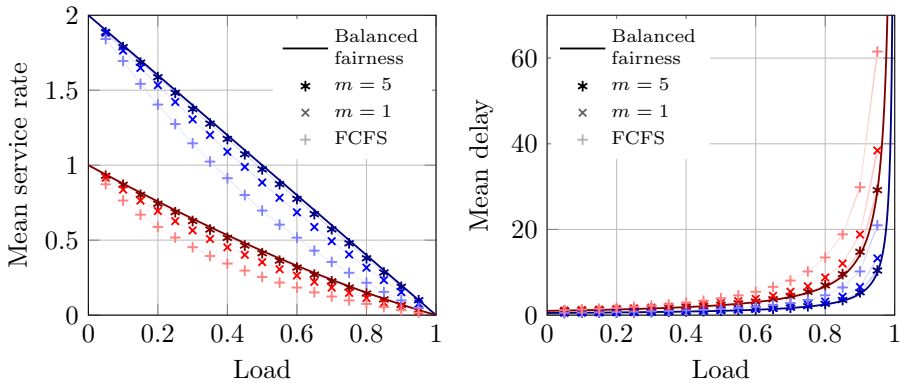


(b) Hyperexponential

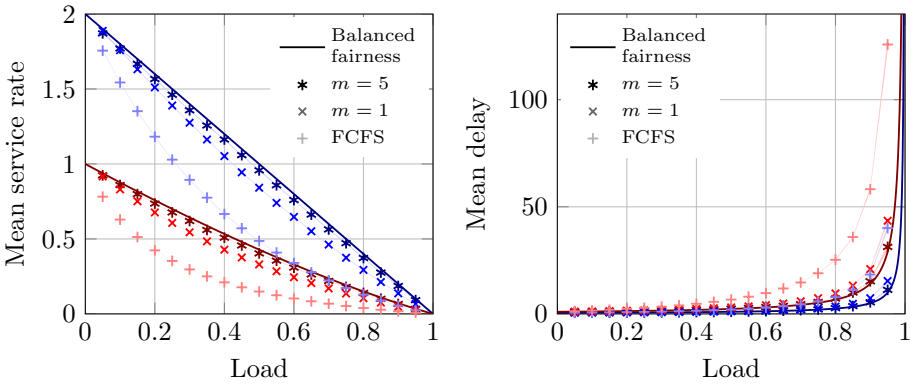


(c) Zipf number of exponentially distributed phases

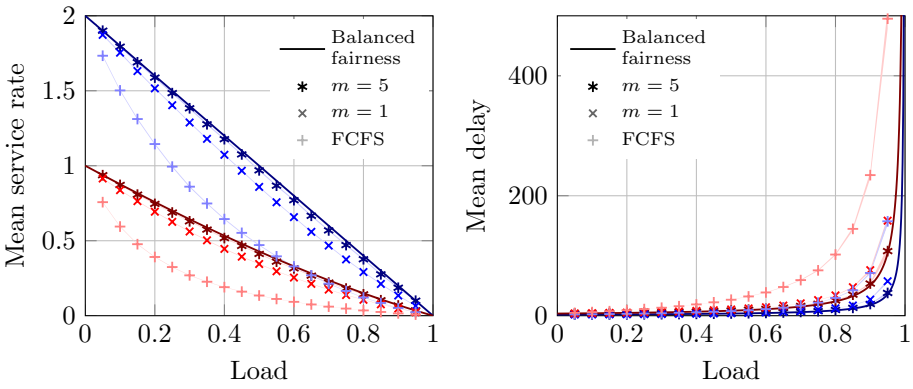
Figure 1: Performance metrics for $N = 2$ job classes sharing $S = 3$ servers ($\mu_1 = \mu_2 = \mu_3 = 1$).



(a) Bimodal number of exponentially distributed phases

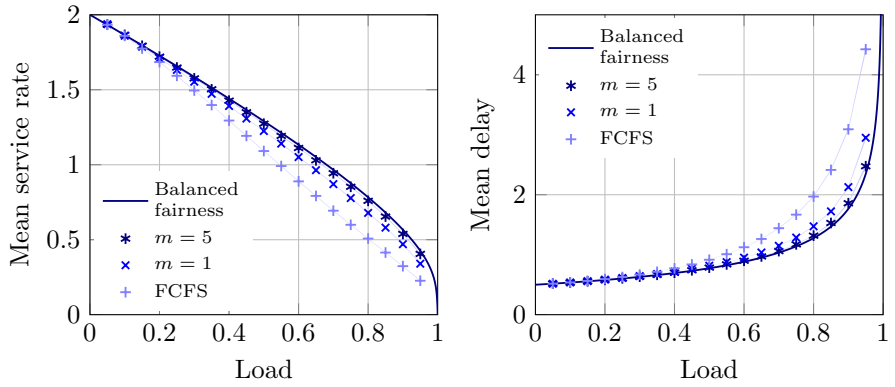


(b) Hyperexponential

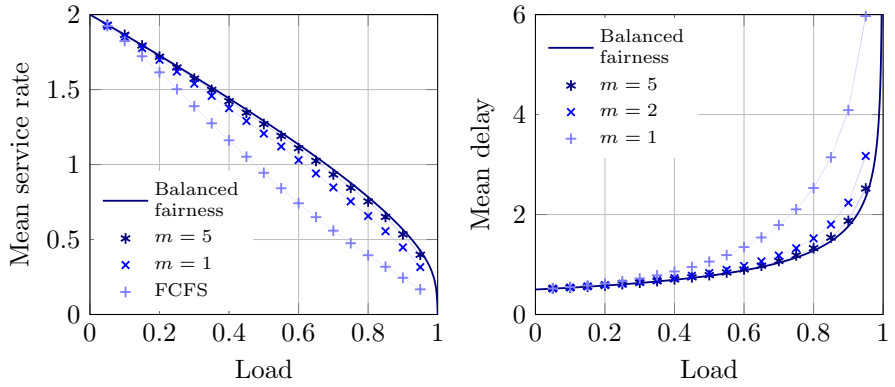


(c) Zipf number of exponentially distributed phases

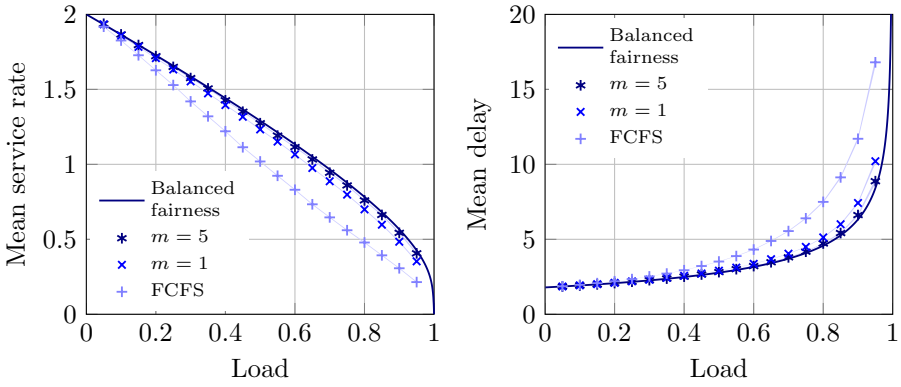
Figure 2: Performance metrics for $N = 2$ job classes sharing $S = 2$ servers ($\mu_1 = \mu_3 = 1$, $\mu_2 = 0$). The performance of class-1 jobs, which have access to both servers, appears in blue on the figure (top plot for the mean service rate and bottom plot for the mean delay). The performance of class-2 jobs, which have access to only one server, appears in red on the figure (bottom plot for the mean service rate and top plot for the mean delay).



(a) Bimodal number of exponentially distributed phases

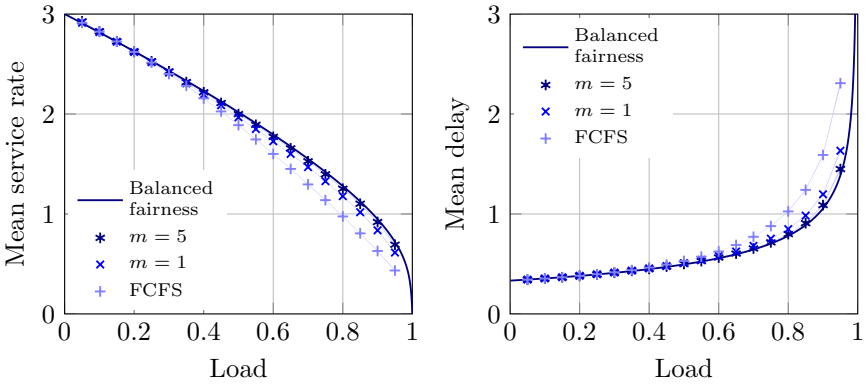


(b) Hyperexponential

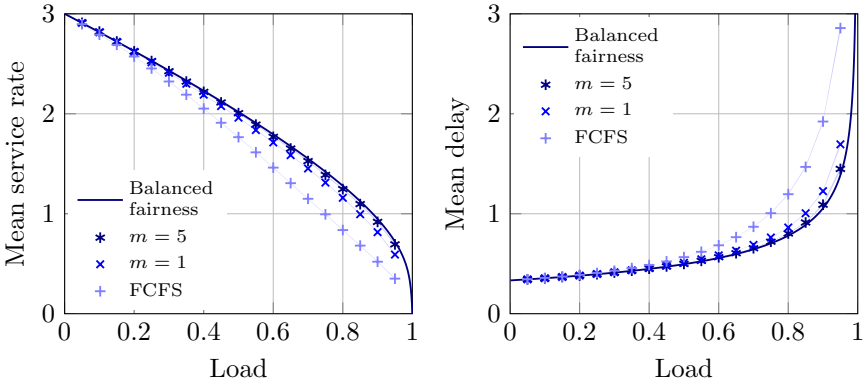


(c) Zipf number of exponentially distributed phases

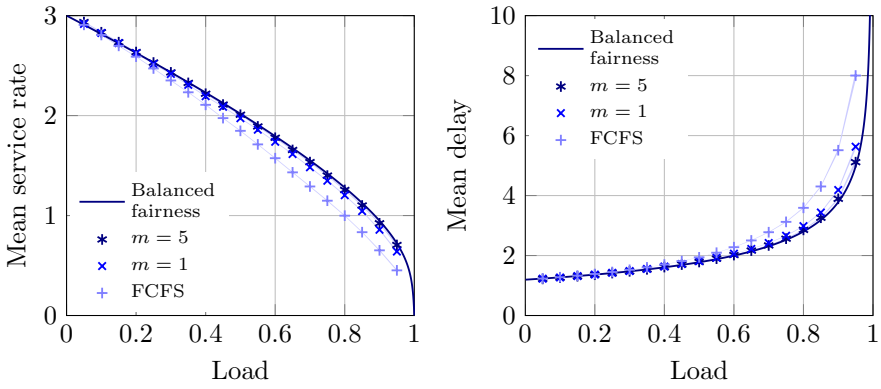
Figure 3: Performance metrics for random assignment of $d = 2$ servers among $S = 100$.



(a) Bimodal number of exponentially distributed phases



(b) Hyperexponential



(c) Zipf number of exponentially distributed phases

Figure 4: Performance metrics for random assignment of $d = 3$ servers among $S = 100$.